# <Shows Management Application>
# Analysis and Design Document

**Student: Plesa Gabriel-Ioan**
**Group: 30235**

# Table of Contents

# 1. Requirements Analysis

### 1.1 Assignment Specification

Use Java/C# API to design and implement a client-server application for managing online show visualization such as movies, theatre performances and sport events. The application has three types of users: the basic user, the premium user and an administrator.

### 1.2 Functional Requirements

The basic user can perform the following operations:

- Search show, select a show and view details of a show
- View history of all shows he has seen
- Give a rating to the show
- Add a comment to the show

The premium user can perform the following operations:

- All operations from basic user
- Recommend a show to a friend or a group of friends who also have accounts on the site and are premium users (the recommendation will also appear as a notification on the friends page)
- Add interests in a show he wants to see when it will be uploaded on the site and receive notification from application that the show was uploaded so that he can watch it

The administrator can perform the following operations:

- CRUD on shows (for ex. movie information: name, description, actors, release date, imdb rating).
- CRUD on user accounts.

In addition, when a new show is uploaded on the site and there are users interested in that show the application should inform all the interested users about that show by sending them an update about the show and let them know they can watch it.

### 1.3 Non-functional Requirements

2 The application should be client-server and the data will be stored in a database.

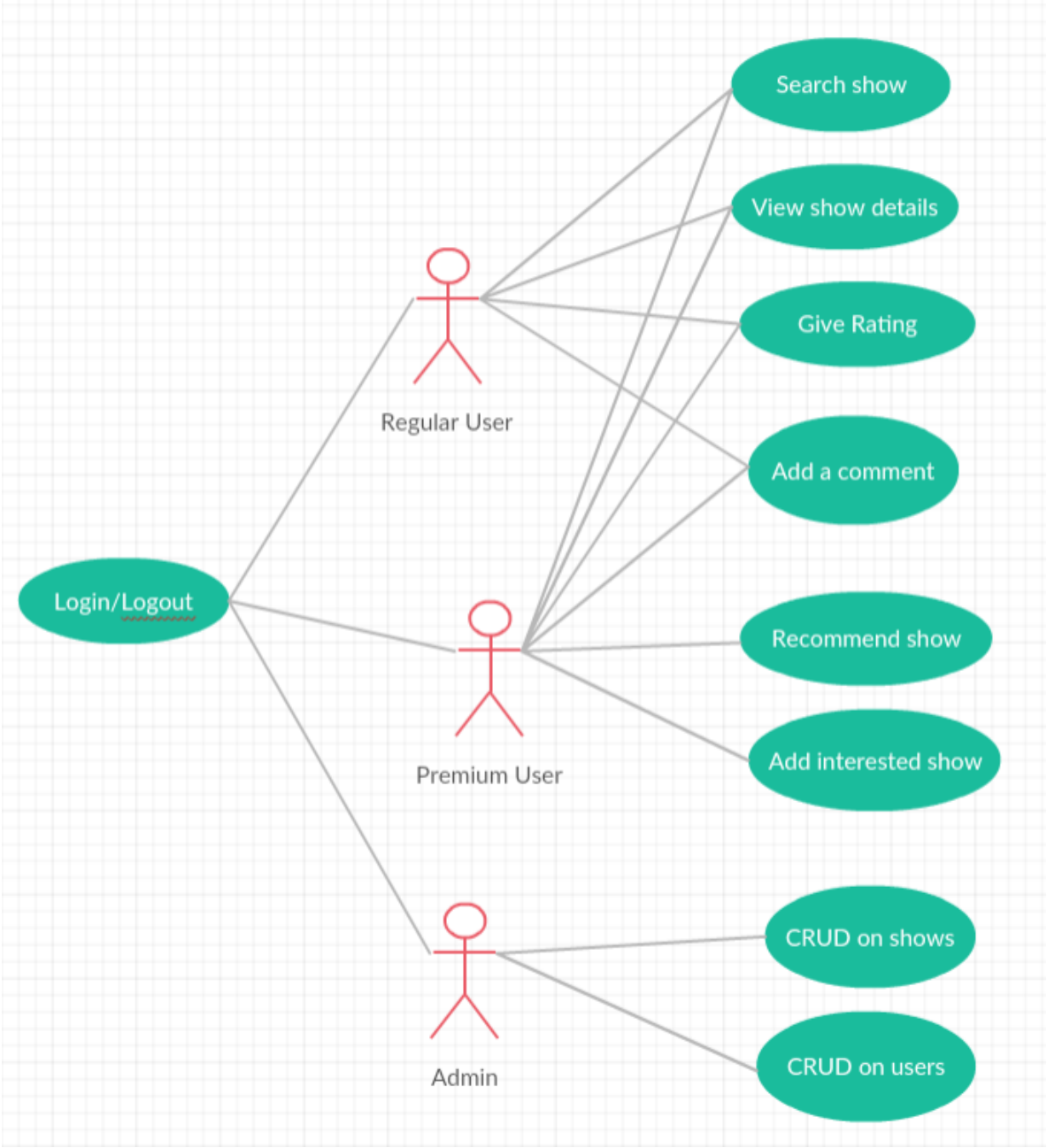
3 Use an ORM (hibernate) for database operations

4 Use the Observer design pattern for notifying the users when the movies they are interested in have been uploaded.

5 Use the bridge design pattern to implement user relationship and show relationship.
(or you can implement a different scenario)

6 All the inputs of the application will be validated against invalid data before submitting the data and saving it.

# 2. Use-Case Model




**Use case:** <Create user>
**Level**: <admin-goal level >
**Primary actor**: <Admin>
**Main success scenario:** <The administrator successfully logins and then he enters the correct data of a user and clicks on the create button then the users is created and table is updated>
**Extensions:** <the administrator wrongly introduce his username or password, an error message is shown, after that he finally login>

# 3. System Architectural Design

## 3.1 Architectural Pattern Description

Model view controller (MVC) pattern is used to separate application's concerns.

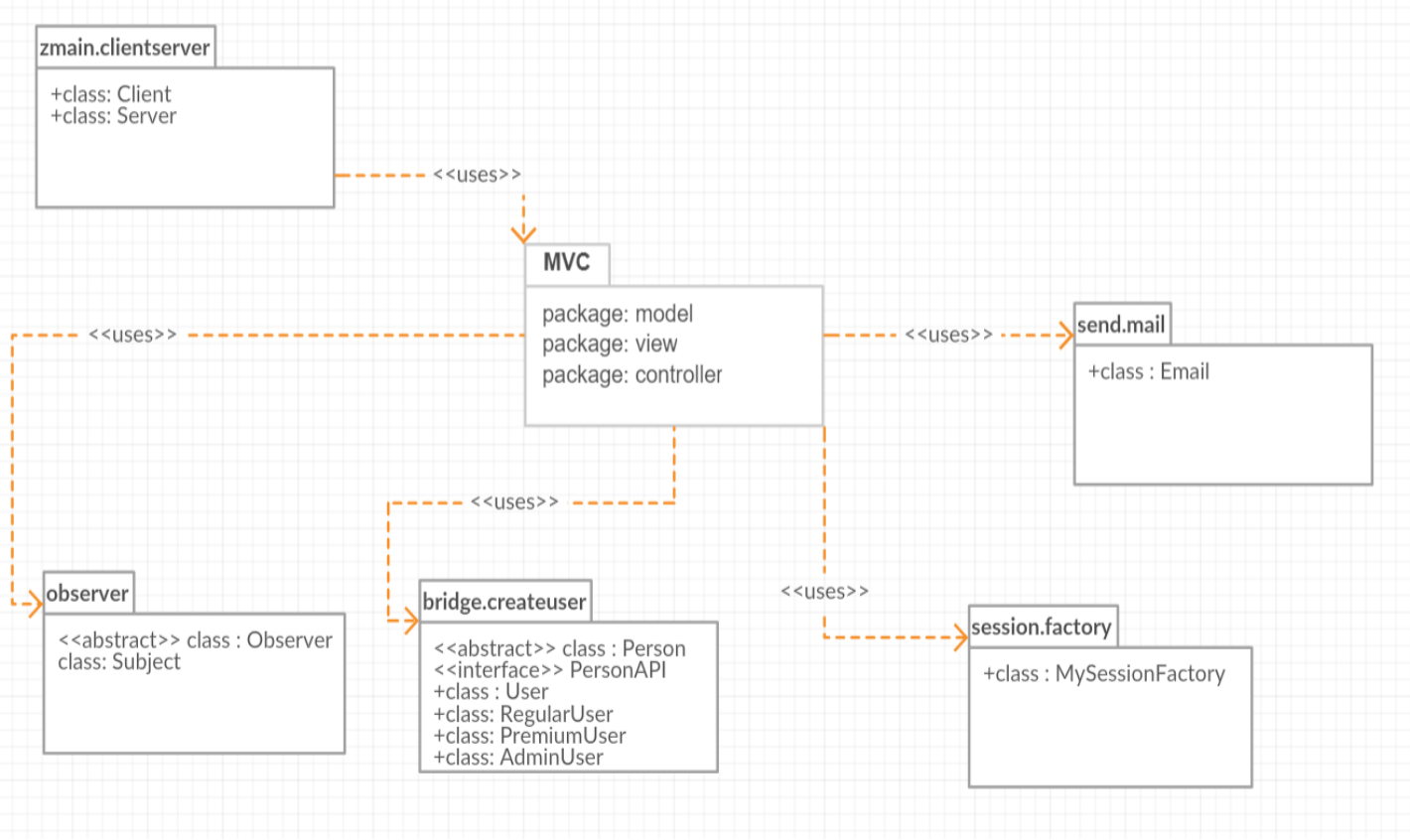
• **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

• **View** - View represents the visualization of the data that model contains.

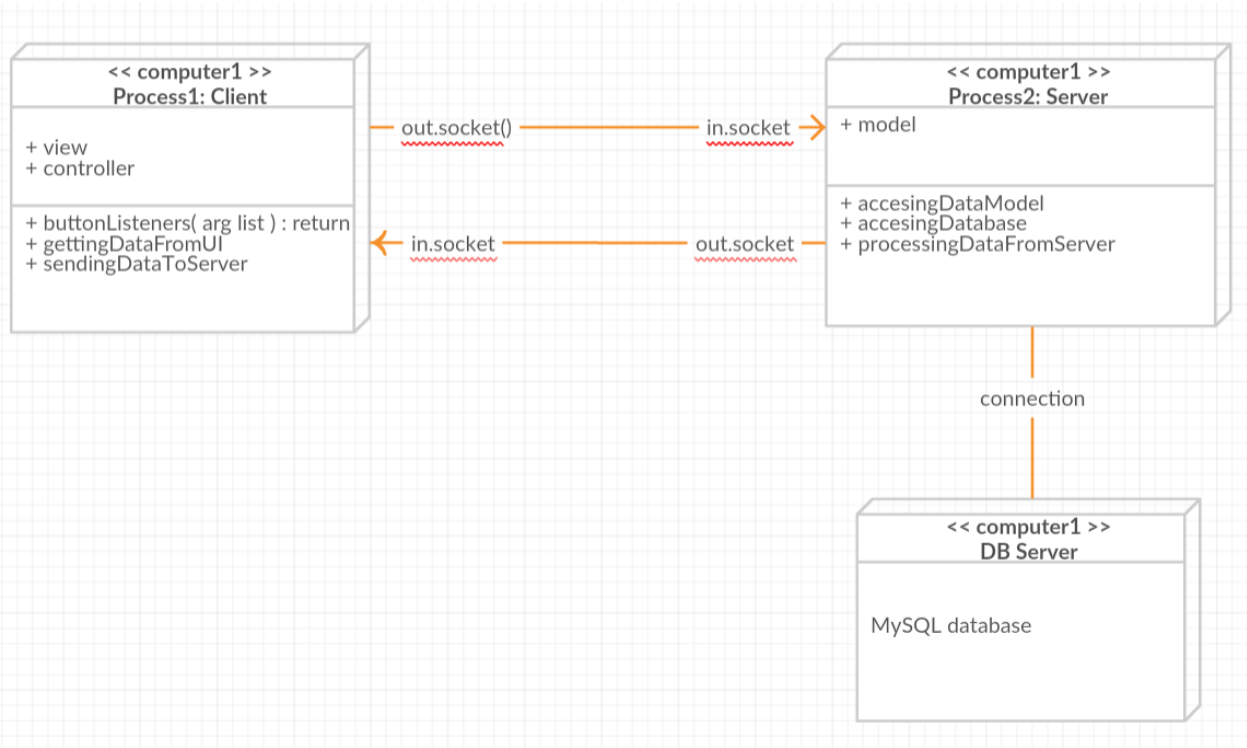
• **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

## 3.2 Diagrams

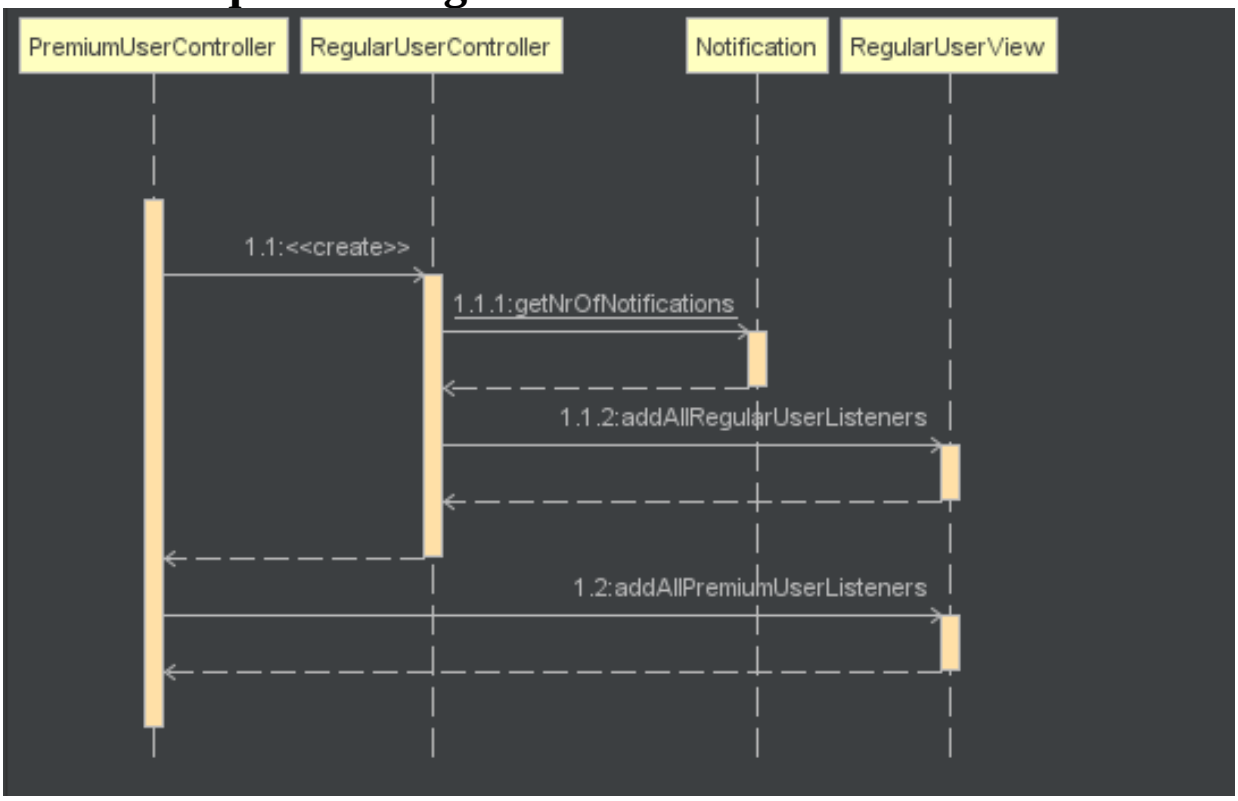
**Package Design**



**Deployment Diagram:**




# 4. UML Sequence Diagrams



# 5. Class Design

## 5.1 Design Patterns Description

1. **Design Patterns - Bridge Pattern** – used in this application to create a user

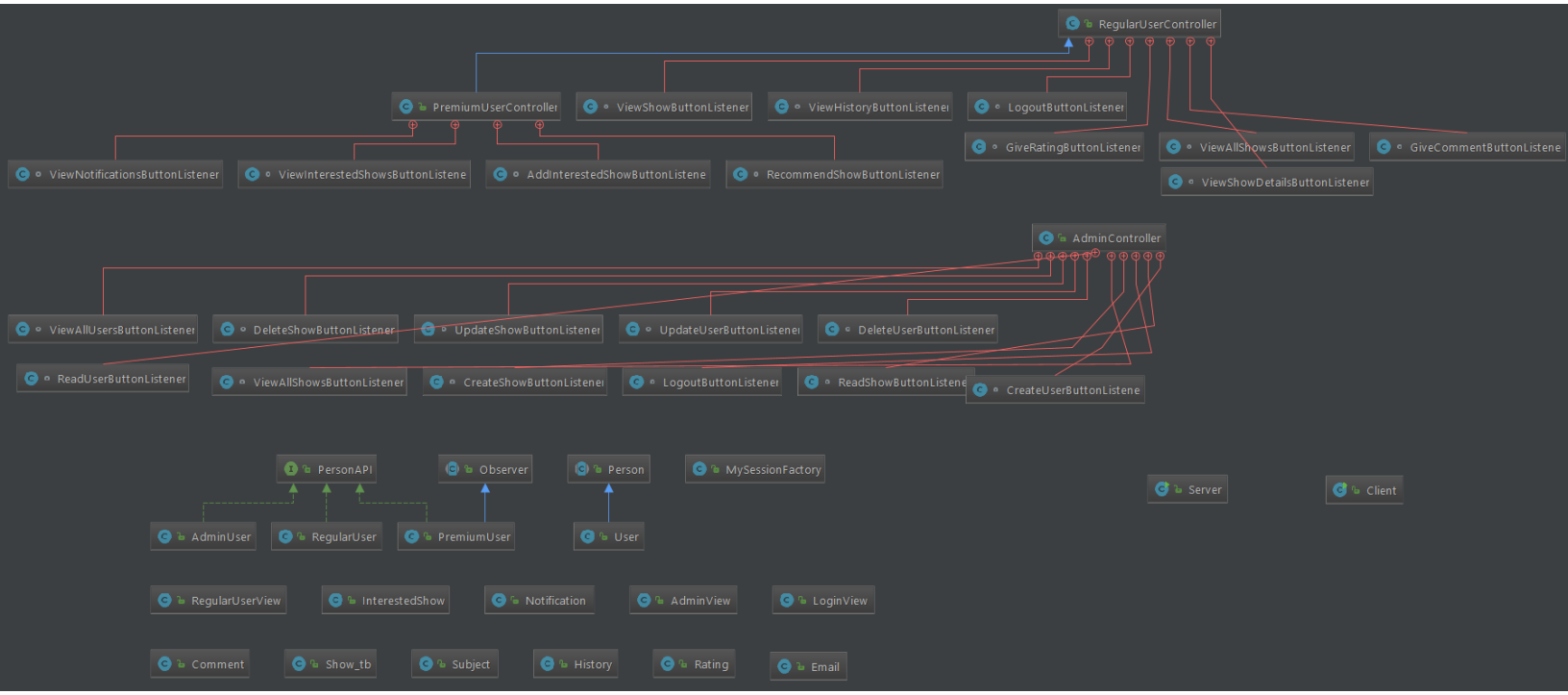
Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

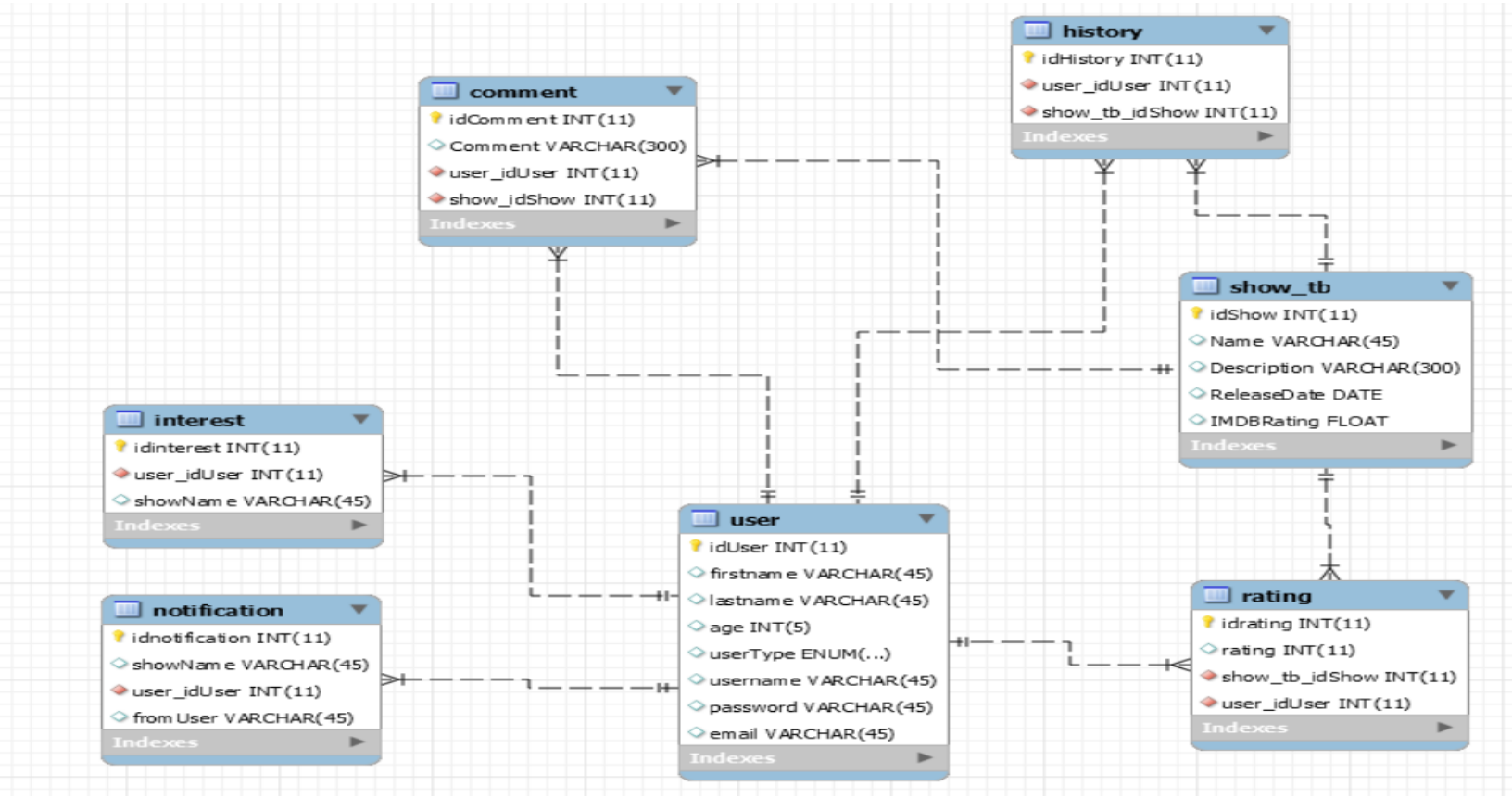
2. **Design Patterns - Observer Pattern** – used to notify a premium user with an e-mail when a show is uploaded on the application

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

## 5.2 UML Class Diagram



# 6. Data Model




# 7. System Testing

For the main operations the system supports tests: insert, delete, update, view, etc. If something is going wrong the application send an error message to inform the user. The information it is tested and validated before it is inserted into Database.

# 8. Bibliography


[1] https://github.com/kittyrad/laborator1
[2] https://www.mkyong.com/java/javamail-api-sending-email-via-gmail-smtp-example/
[3]https://examples.javacodegeeks.com/enterprisejava/hibernate/hibernate-annotations-example/
[4] https://www.tutorialspoint.com/design_pattern/observer_pattern.htm
[5] https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm
[5] www.google.ro